The "Backtracking" assignment.

As usual, comments in RED, input in BLUE and output in BLACK.

To solve a Sudoku puzzle by backtracking.

There are 81 squares arranged in nine rows, nine columns and nine boxes.

Each square lies in one of each. Each digit occurs once in each row, column

and box. Some of the squares start out with digits. The goal is to fill all

the empty squares subject to the above rules. The program will print the

solution or print a message if there is no solution.

General idea: move through the squares. Each row, column and box has nine

boolean variables that are true when a digit is present. False if it isn't.

The "done" condition could be when all 243 boolean variables are true. An easier

"done" condition is when all the squares are filled subject to the rules.

(There should be exactly one choice when each square in the last row is reached.

There cannot be more than that but there may be zero, in which case it needs more processing.)

There are actually a grand total 972 booleans when we count the additional nine

associated with each node that tell us which digits to attempt. If there is more than

one solution this program will stop after the first one it finds. If there is no solution the "getNextSquare" method will return "false" to the main program.

Most squares will be visited more than once. Some of the squares near the beginning may be visited hundreds of times.

Backtracking is a way to do this recursively and try all the possibilities.

A chain of possibilities ends when a square is encountered that has no

possible choice left. Then it backtracks to find the most recent square with an

untried choice. Eventually, the last square will be reached.

The last square is handled separately. If we can fill in a digit we are done, otherwise

backtrack to find another path. The squares are always handled in the same order.

It will take thousands of attempts overall to reach the end. Probably not predictable in advance as

the number would depend on the given nodes and the order that squares are visited in.

One thing that could be tried to speed it up would be, during preprocessing, to

look at each empty square and determine a list of what order to fill them in. I would look

at perhaps the ten with the lowest number of choices to start with. Do them in order of fewest

choices to most, then scan the rest of the list. This could be done with the backtracking algorithm

modified with a "next" method. (If there is a square with only one choice it makes sense to do it first.)

Here is the test run. There are two "canned" boards in the source code. A third is entered from the runstream.

--------------------------------------------------------------------------------------------------------------------------

SudokuTest.main({ ""});

---------------------- // Given problem.

I 56 I 1 I 7 2 I

I 2 I 3 I 154 I

I 3 I 7 I 6 I

----------------------

I I 75 I 94 I

I I 2 6 I I

I 57 I 49 I I

----------------------

I 1 I 8 I 2 I

I 685 I 7 I 1 I

I 3 2 I 4 I 89 I

----------------------

---------------------- // Solution

I 569 I 814 I 732 I

I 728 I 369 I 154 I

I 413 I 752 I 896 I

----------------------

I 236 I 175 I 948 I

I 941 I 286 I 375 I

I 857 I 493 I 621 I

----------------------

I 194 I 538 I 267 I

I 685 I 927 I 413 I

I 372 I 641 I 589 I

----------------------

---------------------- // This problem has no solution.

I 568 I 1 I 7 2 I

I 2 I 3 I 154 I

I 3 I 7 I 6 I

----------------------

I I 75 I 94 I

I I 2 6 I I

I 57 I 49 I I

----------------------

I 1 I 8 I 2 I

I 685 I 7 I 1 I

I 3 2 I 4 I 89 I

----------------------

No solution found

Please input the initial rows, one at a time when prompted.

They should be entered as nine-character Strings with hyphens for the blank squares.

This program does not validate the data

Enter the 1th row now.

9--63---8

Enter the 2th row now.

-----51--

Enter the 3th row now.

-8----7--

Enter the 4th row now.

6----32--

Enter the 5th row now.

-1-----9-

Enter the 6th row now.

--72----3

Enter the 7th row now.

--5----2-

Enter the 8th row now.

--61-----

Enter the 9th row now.

1---79--6

---------------------- // This is the gameboard that was just entered.

I 9 I 63 I 8 I

I I 5 I 1 I

I 8 I I 7 I

----------------------

I 6 I 3 I 2 I

I 1 I I 9 I

I 7 I 2 I 3 I

----------------------

I 5 I I 2 I

I 6 I 1 I I

I 1 I 79 I 6 I

----------------------

---------------------- // This is the solution.

I 971 I 632 I 458 I

I 264 I 785 I 139 I

I 583 I 941 I 762 I

----------------------

I 659 I 413 I 287 I

I 312 I 857 I 694 I

I 847 I 296 I 513 I

----------------------

I 795 I 364 I 821 I

I 436 I 128 I 975 I

I 128 I 579 I 346 I

----------------------

Here is result of the showAll command: // In this case it shows the contents of the "board" array used in the main calculation

1 1 1 true 9 // True if given, false otherwise. Last digit is contents.

2 1 1 false 7

3 1 1 false 1

4 1 2 true 6

5 1 2 true 3

6 1 2 false 2

7 1 3 false 4

8 1 3 false 5

9 1 3 true 8

1 2 1 false 2

2 2 1 false 6

3 2 1 false 4

4 2 2 false 7

5 2 2 false 8

6 2 2 true 5

7 2 3 true 1

8 2 3 false 3

9 2 3 false 9

1 3 1 false 5

2 3 1 true 8

3 3 1 false 3

4 3 2 false 9

5 3 2 false 4

6 3 2 false 1

7 3 3 true 7

8 3 3 false 6

9 3 3 false 2

1 4 4 true 6

2 4 4 false 5

3 4 4 false 9

4 4 5 false 4

5 4 5 false 1

6 4 5 true 3

7 4 6 true 2

8 4 6 false 8

9 4 6 false 7

1 5 4 false 3

2 5 4 true 1

3 5 4 false 2

4 5 5 false 8

5 5 5 false 5

6 5 5 false 7

7 5 6 false 6

8 5 6 true 9

9 5 6 false 4

1 6 4 false 8

2 6 4 false 4

3 6 4 true 7

4 6 5 true 2

5 6 5 false 9

6 6 5 false 6

7 6 6 false 5

8 6 6 false 1

9 6 6 true 3

1 7 7 false 7

2 7 7 false 9

3 7 7 true 5

4 7 8 false 3

5 7 8 false 6

6 7 8 false 4

7 7 9 false 8

8 7 9 true 2

9 7 9 false 1

1 8 7 false 4

2 8 7 false 3

3 8 7 true 6

4 8 8 true 1

5 8 8 false 2

6 8 8 false 8

7 8 9 false 9

8 8 9 false 7

9 8 9 false 5

1 9 7 true 1

2 9 7 false 2

3 9 7 false 8

4 9 8 false 5

5 9 8 true 7

6 9 8 true 9

7 9 9 false 3

8 9 9 false 4

9 9 9 true 6

Here is the descriptor for the Backtracking class.

|  |
| --- |
| Backtracking |
| |  | | --- | | node | | - Character thisDigit // One of the digits 1..9.  - Integer rowNum // The row number of the current node.  - Integer colNum // The column number of the current node  - Integer boxNum // The box number of the current node.  - boolean given // True if given in problem false if unknown. | | - node(int count) // Constructor for node at position count.  + String toString() // Standard toString method. | |
| - node[81] board // The main array for the computation.  - Character[9][9] gameboard // The 9x9 array for inputting and outputting.  - boolean[9][9] row // These three arrays keep track of the filled squares.  - boolean[9][9] column // the first index is the row, column or box number  - boolean[9][9] box // the second index is the digit. True if present.  - Character[9] digits // The digits from 1-9.  - int count // The number of the current square. |
| + Backtracking() // Constructor.  + Character[9][9] getBoard() // Construct and return the current gameboard array.  + boolean getNextSquare(int count) // Recursive, returns true if the current square  and all following squares have the correct entry. (done)  + Character[9][9] inputBoard() // Input a gameboard from the console.  + void printGameboard() // Create a gameboard for the current board array and output it.  + void putBoard(Character[][] gameboard) // Initialize a board array from a gameboard.  + void showAll () // Output the board array. |